
LIBRARY

FILEID**OTSCVTTR

```
  000000    TTTTTTTTTT    SSSSSSSS    CCCCCCCC  VV      VV  TTTTTTTTTT  TTTTTTTTTT  RRRRRRRR
  000000    TTTTTTTTTT    SSSSSSSS    CCCCCCCC  VV      VV  TTTTTTTTTT  TTTTTTTTTT  RRRRRRRR
00      00      TT      SS          CC          VV      VV      TT          TT      RR      RR
00      00      TT      SS          CC          VV      VV      TT          TT      RR      RR
00      00      TT      SS          CC          VV      VV      TT          TT      RR      RR
00      00      TT      SS          CC          VV      VV      TT          TT      RR      RR
00      00      TT        SSSSSS    CC          VV      VV      TT          TT      RRRRRRRR
00      00      TT        SSSSSS    CC          VV      VV      TT          TT      RRRRRRRR
00      00      TT              SS  CC          VV      VV      TT          TT      RR    RR
00      00      TT              SS  CC          VV      VV      TT          TT      RR    RR
00      00      TT              SS  CC            VV  VV        TT          TT      RR      RR      ....
00      00      TT              SS  CC            VV  VV        TT          TT      RR      RR      ....
  000000        TT      SSSSSSSS      CCCCCCCC      VV          TT          TT      RR        RR    ....
  000000        TT      SSSSSSSS      CCCCCCCC      VV          TT          TT      RR        RR    ....

LL              IIIIII      SSSSSSSS
LL              IIIIII      SSSSSSSS
LL                II      SS
LL                II      SS
LL                II      SS
LL                II      SS
LL                II        SSSSSS
LL                II        SSSSSS
LL                II              SS
LL                II              SS
LL                II              SS
LL                II              SS
LLLLLLLLLL      IIIIII    SSSSSSSS
LLLLLLLLLL      IIIIII    SSSSSSSS
```

Table of contents

```
0000     1          .TITLE  OTS$CVTTR       ; Convert text to real (D, G and H)
0000     2          .IDENT  /1-011/                 ; File: OTSCVTTR.MAR  Edit: FM1011
0000     3
0000     4  ;*****************************************************************************
0000     5  ;*                                                                           *
0000     6  ;*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                  *
0000     7  ;*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                   *
0000     8  ;*  ALL RIGHTS RESERVED.                                                     *
0000     9  ;*                                                                           *
0000    10  ;*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED     *
0000    11  ;*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE    *
0000    12  ;*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER     *
0000    13  ;*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY     *
0000    14  ;*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY     *
0000    15  ;*  TRANSFERRED.                                                             *
0000    16  ;*                                                                           *
0000    17  ;*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE     *
0000    18  ;*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT    *
0000    19  ;*  CORPORATION.                                                             *
0000    20  ;*                                                                           *
0000    21  ;*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS     *
0000    22  ;*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                  *
0000    23  ;*                                                                           *
0000    24  ;*                                                                           *
0000    25  ;*****************************************************************************
0000    26  ;
0000    27  ;
0000    28  ; FACILITY: Language-independent support library
0000    29  ;++
0000    30  ; ABSTRACT:
0000    31  ;
0000    32  ; Performs conversion of character strings containing numbers to
0000    33  ; floating datatypes.  This routine supports FORTRAN F, E, D and
0000    34  ; G format conversion, as well as similar types in other languages.
0000    35  ;
0000    36  ;--
0000    37  ;
0000    38  ; VERSION: 1
0000    39  ;
0000    40  ; HISTORY:
0000    41  ;
0000    42  ; AUTHOR:
0000    43  ;       Steven B. Lionel, 2-Jul-79: Version 1
0000    44  ;
0000    45  ;
```

```
                                                J 12
0000     47          .SBTTL  HISTORY          ; Detailed Current Edit History
0000     48
0000     49
0000     50 ; EDIT HISTORY:
0000     51 ;
0000     52 ; 1-001 - Adapted from OTS$CVTTH version 1-003, changed to use Tom
0000     53 ; 1-002 - Add forgotten FOR$CNV_IN_DEFG entry point.  SBL 6-Jul-1979
0000     54 ; 1-003 - Fix bug in SCALE.  SBL 6-Jul-1979
0000     55 ; 1-004 - Use Tom
0000     56 ;         Eggers' multi-precision multiply routine in OTS$$CVTRT.
0000     57 ;         SBL 2-Jul-1979
0000     58 ; 1-005 - Compensate for removal of STRING_LEN from convert frame.
0000     59 ;         SBL 11-Jul-79
0000     60 ; 1-006 - Correct a typo in a comment.  JBS 30-JUL-1979
0000     61 ; 1-007 - Correct implementation of V_SKIPTABS.  SBL 5-Sept-1979
0000     62 ; 1-008 - Implement V_EXP_LETTER.  SBL 4-Dec-1979
0000     63 ; 1-009 - Improve check for overflow, underflow to catch extreme cases.
0000     64 ;         Previously, extreme overflow could give invalid answer with
0000     65 ;         success status.  SBL 17-June-1980
0000     66 ; 1-010 - Speed up operations on FAC when it fits in a longword (9 or
0000     67 ;         fewer digits) or a quadword (18 or fewer digits).  Improve
0000     68 ;         multiplication by 10, test for zero, and normalization.  JAW
0000     69 ;         28-Apr-1981
0000     70 ; 1-011 - The OTS$$CVT_MUL now expects a simpler call interface.  Namely
0000     71 ;         one does not have to find reciprocal of the desired entry in
0000     72 ;         OTS$$A_CVT_TAB to call this routine.  Change the call to
0000     73 ;         OTS$$CVT_MUL to pass the address of the desired entry in
0000     74 ;         OTS$$A_CVT_TAB table instead of its reciprocal.  FM 29-FEB-83
```

```
                     0000     76          .SBTTL  DECLARATIONS
                     0000     77
                     0000     78 ;
                     0000     79 ; INCLUDE FILES:
                     0000     80 ;
                     0000     81
                     0000     82 ;
                     0000     83 ; EXTERNAL SYMBOLS:
                     0000     84 ;
                     0000     85          .DSABL  GBL
                     0000     86          .EXTRN  OTS$_INPCONERR          ; Input conversion error
                     0000     87          .EXTRN  OTS$$A_CVT_TAB          ; Convert table address
                     0000     88          .EXTRN  OTS$$CVT_MUL            ; Conversion multiply routine
                     0000     89
                     0000     90 ;
                     0000     91 ; MACROS:
                     0000     92 ;
                     0000     93
                     0000     94 ;
                     0000     95 ; PSECT DECLARATIONS:
                     0000     96 ;
                     0000     97
                00000000     98          .PSECT  _OTS$CODE       PIC, SHR, LONG, EXE, NOWRT
                     0000     99
                     0000    100 ;
                     0000    101 ; EQUATED SYMBOLS:
                     0000    102 ;
                     0000    103
                     0000    104
                000003FC     0000    105          REGMASK         = ^M<R2, R3, R4, R5, R6, R7, R8, R9>
                     0000    106                                          ; register save mask
                     0000    107                                          ; Note: integer overflow not enabled
                     0000    108
                     0000    109 ;+
                     0000    110 ; The following symbols are used to indicate the bit position of the flag
                     0000    111 ; register.
                     0000    112 ;-
                     0000    113
                0000001F     0000    114          V_NEGATIVE      = 31            ; flag bit: 1 if negative sign
                0000001E     0000    115          V_DEC_POINT     = 30            ; flag bit: 1 if decimal point is seen
                40000000     0000    116          M_DEC_POINT     = 1@30          ; mask for V_DEC_POINT
                0000001D     0000    117          V_NEG_DECEXP    = 29            ; flag bit: T if exponent has negative sign
                20000000     0000    118          M_NEG_DECEXP    = 1@29          ; mask for V_NEG_DEXEXP
                0000001C     0000    119          V_DECEXP        = 28            ; flag bit: T if exponent field exist
                10000000     0000    120          M_DECEXP        = 1@28          ; mask for V_DECEXP
                0000001B     0000    121          V_EXT_BITS      = 27            ; flag bit: T if extension bits
                     0000    122                                                  ; wanted
                08000000     0000    123          M_EXT_BITS      = 1@27          ; mask for V_EXT_BITS
                     0000    124
                     0000    125
                     0000    126 ;+
                     0000    127 ; Literals for data types
                     0000    128 ;-
                00000000     0000    129          K_DTYPE_D       = 0             ; D-floating
                00000001     0000    130          K_DTYPE_G       = 1             ; G-floating
                00000002     0000    131          K_DTYPE_H       = 2             ; H-floating
                     0000    132
```

```
                     0000   133 ;+
                     0000   134 ; Temporary stack offsets
                     0000   135 ;-
                     0000   136
00000000             0000   137         TEMP            = 0             ; temporary storage during
                     0000   138                                         ; 8 word shift
00000004             0000   139         FLAG            = 4             ; flag storage
                     0000   140                                         ; was R6 in FOR$CNV_IN_DEFG
00000008             0000   141         DIGITS          = 8             ; digits to right of decimal
                     0000   142                                         ; point (was R7)
0000000C             0000   143         DECEXP          = 12            ; Decimal exponent
00000010             0000   144         DTYPE           = 16            ; Datatype code
                     0000   145
                     0000   146 ;+
                     0000   147 ; Stack offsets for OTS$$CVT_MUL routine
                     0000   148 ;-
00000014             0000   149         BINNUM          = 20            ; Binary fraction storage
00000024             0000   150         INT             = 36            ; Overflow area for BINNUM
00000028             0000   151         BINEXP          = 40            ; Binary exponent
0000002C             0000   152         PRODF_4         = 44            ; Multiply temporary
00000030             0000   153         PRODF           = 48            ; Multiply temporary
00000040             0000   154         CRY             = 64            ; Carry save area
00000050             0000   155         FRAME           = CRY + 16      ; Stack frame size
                     0000   156
                     0000   157 ;+
                     0000   158 ; Constants
                     0000   159 ;-
                     0000   160
0CCCCCCC             0000   161         L_2P31_DIV_10   = 214748364     ; (2**31)/10
                     0000   162
```

```
0000   164          .SBTTL  OTS$CVT_T_x - convert text to floating
0000   165
0000   166 ;++
0000   167 ; FUNCTIONAL DESCRIPTION:
0000   168 ;
0000   169 ;       OTS$CVT_T_x converts a text string containing a representation
0000   170 ;       of a numeric value to a floating representation of that
0000   171 ;       value.  The routine supports FORTRAN F,E,D and G input type
0000   172 ;       conversion as well as similar types for other languages.
0000   173 ;
0000   174 ;       The description of the text representation converted by
0000   175 ;       OTS$CVT_T_x is as follows:
0000   176 ;
0000   177 ;               <0 or more blanks>
0000   178 ;               <"+", "-" or nothing>
0000   179 ;               <0 or more decimal digits>
0000   180 ;               <"." or nothing>
0000   181 ;               <0 or more decimal digits>
0000   182 ;               <exponent or nothing, where exponent is:
0000   183 ;                       <       <<"E", "e", "D", "d", "Q", "q">
0000   184 ;                               <0 or more blanks>
0000   185 ;                               <"+", "-" or nothing>>
0000   186 ;                               or
0000   187 ;                               <"+" or "-">>
0000   188 ;                       <0 or more decimal digits>>
0000   189 ;               <end of string>
0000   190 ;
0000   191 ;               Notes:  1. Unless "caller_flags" bit V_SKIPBLANKS
0000   192 ;                          is set, blanks are equivalent to
0000   193 ;                          decimal "0".  If V_SKIPBLANKS is set,
0000   194 ;                          blanks are always ignored.
0000   195 ;                       2. There is no difference in semantics
0000   196 ;                          between any of the 6 valid exponent
0000   197 ;                          letters.
0000   198 ;                       3. If "caller_flags" bit V_ONLY_E is set,
0000   199 ;                          the only valid exponent letters are
0000   200 ;                          "E" and "e"; any others will be treated
0000   201 ;                          as an invalid character.
0000   202 ;                       4. If "caller_flags" bit V_SKIPTABS is set,
0000   203 ;                          tab characters are ignored else they are
0000   204 ;                          an error.
0000   205 ;                       5. If "caller_flags" bit V_EXP_LETTER is set,
0000   206 ;                          the exponent, if present, must start with
0000   207 ;                          a valid exponent letter, i.e. 1.2E32.
0000   208 ;                          If clear, the exponent letter may be omitted,
0000   209 ;                          i.e. 1.2+32.
0000   210 ;
0000   211 ;
0000   212 ; CALLING SEQUENCE:
0000   213 ;
0000   214 ;       status.wlc.v = OTS$CVT_T_x (in_str.rt.dx, value.wfx.r
0000   215 ;                                  [, digits_in_fract.rlu.v
0000   216 ;                                  [, scale_factor.rl.v
0000   217 ;                                  [, caller_flags.rlu.v,
0000   218 ;                                  [, ext_bits.wx.r]]]])
0000   219 ;
0000   220 ;       where "x" is the datatype of the floating value, either
```

```
                     0000   221 ;       D, G or H.
                     0000   222 ;
                     0000   223 ;
                     0000   224 ; INPUT PARAMETERS:
                     0000   225 ;
00000004             0000   226         in_str          = 4             ; input string descriptor by
                     0000   227                                         ; reference.
0000000C             0000   228         digits_in_fract = 12            ; If no decimal point is
                     0000   229                                         ; present in input, specifies
                     0000   230                                         ; how many digits are to be
                     0000   231                                         ; treated as being to the
                     0000   232                                         ; right of the decimal point.
                     0000   233                                         ; If omitted, 0 is the default.
00000010             0000   234         scale_factor    = 16            ; signed scale factor.  If
                     0000   235                                         ; present, and exponent absent,
                     0000   236                                         ; the result value is
                     0000   237                                         ; multiplied by 10**factor.
                     0000   238                                         ; If "caller flags" bit
                     0000   239                                         ; V_FORCESCALE is on, the
                     0000   240                                         ; scale factor is always applied.
00000014             0000   241         caller_flags    = 20            ; flags supplied by caller
                     0000   242     ;+
                     0000   243     ; Definitions of caller supplied flags
                     0000   244     ;-
                     0000   245
00000000             0000   246         V_SKIPBLANKS    = 0             ; If set, blanks are ignored
00000001             0000   247         V_ONLY_E        = 1             ; If set, only E or e exponents
                     0000   248                                         ; allowed (BASIC+2, PL/I)
00000002             0000   249         V_ERR_UFLO      = 2             ; If set, error on underflow
00000003             0000   250         V_DONTROUND     = 3             ; If set, don't round value
00000008             0000   251         M_DONTROUND     = 1@3           ; Mask for V_DONTROUND
00000004             0000   252         V_SKIPTABS      = 4             ; If set, tabs are ignored.
                     0000   253                                         ; If clear, tabs are illegal.
00000005             0000   254         V_EXP_LETTER    = 5             ; If set, an exponent must begin
                     0000   255                                         ; with a valid exponent letter.
                     0000   256                                         ; If clear, the exponent letter
                     0000   257                                         ; may be omitted.
00000006             0000   258         V_FORCESCALE    = 6             ; If set, the scale factor is
                     0000   259                                         ; always applied.  If clear, it
                     0000   260                                         ; is only applied if there is
                     0000   261                                         ; no exponent present in the
                     0000   262                                         ; string.
                     0000   263
00000007             0000   264         NO_OF_FLAGS     = 7             ; Number of flags
                     0000   265
                     0000   266 ;
                     0000   267 ; IMPLICIT INPUTS:
                     0000   268 ;
                     0000   269 ;
                     0000   270 ;       NONE
                     0000   271 ;
                     0000   272 ; OUTPUT PARAMETERS:
                     0000   273 ;
00000008             0000   274         value           = 8             ; floating result by ref
00000018             0000   275         ext_bits        = 24            ; If present, the value will
                     0000   276                                         ; NOT be rounded and the first
                     0000   277                                         ; n bits after truncation wil
```

```
                                                                ; be returned in this argument.
                                                                ; For D-floating, the next 8 bits
                                                                ; are returned as a byte.
                                                                ; For G and H floating, 11 and 15
                                                                ; bits are returned, respectively,
                                                                ; as a word, left-adjusted.
                                                                ; These values are suitable for
                                                                ; use as the extension operand
                                                                ; in an EMOD instruction.
                                                                ; WARNING:  The bits returned for
                                                                ; H-floating may not be precise,
                                                                ; due to the fact that calculations
                                                                ; are only carried to 128 bits.
                                                                ; However, the error should be
                                                                ; small.  D and G datatypes
                                                                ; return guaranteed exact bits,
                                                                ; but they are not rounded.
                                                        ;
                                                        ; IMPLICIT OUTPUTS:
                                                        ;
                                                        ;       NONE
                                                        ;
                                                        ; COMPLETION CODES:
                                                        ;
                                                        ;       OTS$_INPCONERR  - Error if illegal character in input or
                                                        ;                         overflow.
                                                        ;       SS$_NORMAL      - success
                                                        ;
                                                        ; SIDE EFFECTS:
                                                        ;
                                                        ;       NONE
                                                        ;
                                                        ;--

                                                        ;
                                                        ;
                                                        ;
                                03FC    0000            .ENTRY  OTS$CVT_T_H, REGMASK
                                        0002                                    ; entry for OTS$CVT_T_H
5E      00000050 8F     C2      0002            SUBL2   #FRAME, SP              ; Create stack frame
        10 AE    02     D0      0009            MOVL    #K_DTYPE_H, DTYPE(SP)   ; Set datatype code
                 1C     11      000D            BRB     COMMON                  ; Go to common code
                                000F
                                03FC    000F    .ENTRY  OTS$CVT_T_G, REGMASK
                                        0011                                    ; entry for OTS$CVT_T_G
5E      00000050 8F     C2      0011            SUBL2   #FRAME, SP              ; Create stack frame
        10 AE    01     D0      0018            MOVL    #K_DTYPE_G, DTYPE(SP)   ; Set datatype code
                 0D     11      001C            BRB     COMMON                  ; Go to common code
                                001E
                                001E    FOR$CNV_IN_DEFG::
                                03FC    001E    .ENTRY  OTS$CVT_T_D, REGMASK
5E      00000050 8F     C2      0020            SUBL2   #FRAME, SP              ; Create stack frame
        10 AE    00     D0      0027            MOVL    #K_DTYPE_D, DTYPE(SP)   ; Set datatype code
                                002B    ;       BRB     COMMON                  ; Go to common code
                                002B
                                002B    ;+
                                002B    ;       Register usage and abbreviations:
```

```
                                002B   335 ;
                                002B   336 ;       R0 - Generally count of input characters remaining.
                                002B   337 ;       R1 - Generally pointer to input character.
                                002B   338 ;       R2 - Generally holds decimal exponent.
                                002B   339 ;       R3 - Used first to hold current character, then as
                                002B   340 ;            extra precision bits for the fraction.
                                002B   341 ;       R4-R7 - The 128 bit binary fraction.
                                002B   342 ;       R8 - Count of digits seen after overflow.
                                002B   343 ;       R9 - Count of significant digits seen in fraction (number of
                                002B   344 ;             digits currently held in R4:R7).
                                002B   345 ;
                                002B   346 ;       FAC: Binary fraction, R4-R7.
                                002B   347 ;-
                                002B   348
                                002B   349 COMMON:
                          04 AE  D4   002B   350           CLRL    FLAG(SP)                        ; clear flags
                       05   6C  91   002E   351           CMPB    (AP), #<caller_flags/4>         ; is optional caller_flags
                                      0031   352                                                   ; argument present?
                            14  1F   0031   353           BLSSU   5$                              ; if not, skip
      04 AE   07   00       14  AC   FO   0033   354           INSV    caller_flags(AP), #0, #NO_OF_FLAGS, FLAG(SP)
                                      003A   355                                                   ; set caller flags
                       06   6C  91   003A   356           CMPB    (AP), #<ext_bits/4>             ; is optional ext_bits argument
                                      003D   357                                                   ; present?
                            08  1F   003D   358           BLSSU   5$                              ; if not, skip
            04 AE    08000008  8F   CB   003F   359           BISL    #<M_EXT_BITS+M_DONTROUND>, FLAG(SP)
                                      0047   360                                                   ; set bit indicating it is there
                                      0047   361                                                   ; plus dont round bit
                    50   04  BC   7D   0047   362 5$:       MOVQ    @in_str(AP), R0                 ; R0 will get string length, the
                                      004B   363                                                   ; CLASS and TYPE fields will go
                                      004B   364                                                   ; away after the first SKPC.
                                      004B   365                                                   ; R1 points to input string.
                        52  D4   004B   366           CLRL    R2                              ; R2 = DECIMAL_EXPONENT = 0
                        54  7C   004D   367           CLRQ    R4                              ; R4-R7 = FAC = 0
                        56  7C   004F   368           CLRQ    R6
                     08 AE  D4   0051   369           CLRL    DIGITS(SP)                      ; digits in fraction
                  03    6C  91   0054   370           CMPB    (AP), #<digits_in_fract/4>
                                      0057   371                                                   ; is digits_in_fract present?
                        05  1F   0057   372           BLSSU   10$                             ; skip if not
      08 AE    0C  AC   D0   0059   373           MOVL    digits_in_fract(AP), DIGITS(SP) ; set if present
                        58  7C   005E   374 10$:     CLRQ    R8                              ; Clear digit counts (R8 & R9).
                                      0060   375
```

```
                                0060   377 ;+
                                0060   378 ; Find first non-blank.  If none, return zero.  Otherwise process
                                0060   379 ; character.
                                0060   380 ;-
                                0060   381
      61    50    20    3B      0060   382 20$:    SKPC    #^A/ /, R0, (R1)            ; skip blanks
                                0064   383                                            ; R0 = #CHAR_REMAINING
                                0064   384                                            ; R1 = POINTER_TO_INPUT
                                0064   385                                            ; Z bit is set if all blanks
                  03    14      0064   386          BGTR    30$                       ; non-blank found?
                0011C   31      0066   387          BRW     ZERO                      ; if not, return zero
            53    61    9A      0069   388 30$:    MOVZBL  (R1), R3                  ; R3 = ASCII(current_char)
   0D 04 AE       04    E1      006C   389          BBC     #V_SKIPTABS, FLAG(SP), 35$  ; Not skipping tabs?
                  09    53      D1     0071   390          CMPL    R3, #9            ; Is character a tab?
                        08      12     0074   391          BNEQ    35$               ; No
                        51      D6     0076   392          INCL    R1                ; Yes, bump pointer
                     E5 50      F5     0078   393          SOBGTR  R0, 20$           ; Decrement character count
                   0107 31      007B   394          BRW     ZERO                      ; Value is zero
                  2D    53      91     007E   395 35$:    CMPB    R3, #^A/-/        ; is current char a "-" sign?
                        05      12     0081   396          BNEQ    40$               ; branch if not
      15 04 AE          1F      E3     0083   397          BBCS    #V_NEGATIVE, FLAG(SP), DIGIT_LOOP
                                0088   398                                            ; set negative flag and continue
                  2B    53      91     0088   399 40$:    CMPB    R3, #^A/+/        ; is current char a "+" sign?
                        10      13     008B   400          BEQL    DIGIT_LOOP        ; yes, ignore and continue
                  2E    53      91     008D   401          CMPB    R3, #^A/./        ; is current char a "."?
                        15      12     0090   402          BNEQ    CHECK_DIGIT       ; no, should be a digit
   04 AE    40000000    8F      C8     0092   403          BISL    #M_DEC_POINT, FLAG(SP)  ; set decimal point encountered
                  08 AE D4      009A   404          CLRL    DIGITS(SP)                ; ignore digits_in_fract
                                009D   405
```

```
                                009D   407 ;+
                                009D   408 ; Collect integer and fraction digits.  Blanks are zeroes unless
                                009D   409 ; V_SKIPBLANKS is set in which case they are ignored.
                                009D   410 ; Tabs are illegal unless V_SKIPTABS is on in which case they are ignored.
                                009D   411 ;-
                                009D   412
                                009D   413 DIGIT_LOOP:
                     032B   30  009D   414         BSBW    RGET                    ; get a new character
                       50   D5  00A0   415         TSTL    R0                      ; check for end of string
                       03   14  00A2   416         BGTR    CHECK_DIGIT             ; continue if positive
                     00DA   31  00A4   417         BRW     SCALE                   ; done if string empty
                                00A7   418 CHECK_DIGIT:
                   53   30   C2  00A7   419         SUBL    #^A/0/, R3              ; convert to numeric
                   09   53   D1  00AA   420         CMPL    R3, #9                  ; is it a digit?
                     1A   1A   00AD   421         BGTRU   NOT_DIGIT               ; no
           0CCCCCCC 8F   57   D1  00AF   422         CMPL    R7, #L_2P31_DIV_10      ; check highest part of FAC to
                                00B6   423                                         ; see if it is too big to
                                00B6   424                                         ; multiply by 10.
                       04   1B  00B6   425         BLEQU   10$                     ; it's ok
                       58   D6  00B8   426         INCL    R8                      ; overflow, bump counter
                       03   11  00BA   427         BRB     2$                      ; skip multiplication
                     032D   30  00BC   428 10$:    BSBW    MUL10_R9                ; Multiply FAC by 10 and add R3.
                D9 04 AE   1E   E1  00BF   429 2$:     BBC     #V_DEC_POINT, FLAG(SP), DIGIT_LOOP
                                00C4   430                                         ; check to see if decimal
                                00C4   431                                         ; point has been seen
                                00C4   432                                         ; - continue if not.
                   08 AE   D6  00C4   433         INCL    DIGITS(SP)              ; bump DIGITS
                     D4   11  00C7   434         BRB     DIGIT_LOOP              ; branch back to read more
                                00C9   435
```

```
                                00C9   437 ;+
                                00C9   438 ; A non-digit has been found.  Check for sign or exponent letter.
                                00C9   439 ;-
                                00C9   440
                                00C9   441 NOT_DIGIT:
FFFFFFFE 8F    53    D1         00C9   442         CMPL    R3, #<^A/./-^A/0/>      ; check if current char is a "."
                4C    13         00D0   443         BEQL    DECIMAL_POINT           ; branch to DECIMAL_POINT if yes
FFFFFFFB 8F    53    D1         00D2   444         CMPL    R3, #<^A/+/-^A/0/>      ; "+"?
                33    13         00D9   445         BEQL    EXP_PLUS                ; Exponent starts with plus
FFFFFFFD 8F    53    D1         00DB   446         CMPL    R3, #<^A/-/-^A/0/>      ; "-"?
                32    13         00E2   447         BEQL    EXP_MINUS               ; Exponent starts with a minus
          15    53    D1         00E4   448         CMPL    R3, #<^A/E/-^A/0/>      ; "E"?
                40    13         00E7   449         BEQL    EXPON                   ; process exponent
          35    53    D1         00E9   450         CMPL    R3, #<^A/e/-^A/0/>      ; "e"?
                3B    13         00EC   451         BEQL    EXPON                   ; process exponent
       18 04 AE    01    E0      00EE   452         BBS     #V_ONLY_E, FLAG(SP), 10$
                                00F3   453                                         ;       ERROR
                                                                                    ; error if only E, e allowed
          14    53    D1         00F3   454         CMPL    R3, #<^A/D/-^A/0/>      ; "D"?
                31    13         00F6   455         BEQL    EXPON                   ; process exponent
          34    53    D1         00F8   456         CMPL    R3, #<^A/d/-^A/0/>      ; "d"?
                2C    13         00FB   457         BEQL    EXPON                   ; process exponent
          21    53    D1         00FD   458         CMPL    R3, #<^A/Q/-^A/0/>      ; "Q"?
                27    13         0100   459         BEQL    EXPON                   ; process exponent
00000041 8F    53    D1         0102   460         CMPL    R3, #<^A/q/-^A/0/>      ; "q"?
                1E    13         0109   461         BEQL    EXPON                   ; process exponent
              0089    31         010B   462 10$:    BRW     ERROR                   ; error since illegal char.
                                010E   463
                                010E   464 ;+
                                010E   465 ; The exponent did not start with a letter.  This is not allowed
                                010E   466 ; if V_EXP_LETTER is set.
                                010E   467 ;-
                                010E   468 EXP_PLUS:
41 04 AE     05    E1         010E   469         BBC     #V_EXP_LETTER, FLAG(SP), EXP_LOOP
              0081    31         0113   470         BRW     ERROR                   ; Not allowed
                                0116   471 EXP_MINUS:
7C 04 AE     05    E0         0116   472         BBS     #V_EXP_LETTER, FLAG(SP), ERROR
              002E    31         C11B   473         BRW     EXP_NEG                 ; Ok
                                011E   474 ;+
                                011E   475 ; Decimal point has been found
                                011E   476 ;-
                                011E   477
                                011E   478 DECIMAL_POINT:
74 04 AE     1E    E2         011E   479         BBSS    #V_DEC_POINT, FLAG(SP), ERROR   ; error if duplicate
           08 AE    D4         0123   480         CLRL    DIGITS(SP)              ; reset DIGITS
              FF74    31         0126   481         BRW     DIGIT_LOOP              ; get fraction digits
```

```
                                0129   483 ;+
                                0129   484 ; Loop to collect digits, store the accumulated DECIMAL_EXPONENT in R2
                                0129   485 ;-
                                0129   486
                                0129   487 EXPON:
                          50    D7   0129   488          DECL    R0                              ; skip over letter
                          44    15   012B   489          BLEQ    EXP_DONE                        ; done if string empty
                          51    D6   012D   490          INCL    R1                              ; R1 points to next character
                    61    50    20   38   012F   491          SKPC    #^A/ /, R0, (R1)                ; skip blanks
                          3C    15   0133   492          BLEQ    EXP_DONE                        ; done if end of string
                          53    61   9A   0135   493          MOVZBL  (R1), R3                ; R3 = current char
                05 04 AE    04    E1   0138   494          BBC     #V_SKIPTABS, FLAG(SP), 10$      ; Not skipping tabs?
                          09    53   D1   013D   495          CMPL    R3, #9                  ; Is it a tab?
                          E7    13   0140   496          BEQL    EXPON                           ; Yes, skip it
                          2B    53   D1   0142   497 10$:     CMPL    R3, #^A/+/              ; "+"?
                          0D    13   0145   498          BEQL    EXP_LOOP                        ; yes, get digits
                          2D    53   D1   0147   499          CMPL    R3, #^A/-/              ; "-"?
                          0F    12   014A   500          BNEQ    EXP_CHECK                       ; no, go check digit
                                0154   501 EXP_NEG:
            04 AE    20000000   BF   C8   014C   502          BISL    #M_NEG_DECEXP, FLAG(SP) ; exponent is negative
                                0154   503 EXP_LOOP:
                        0274    30   0154   504          BSBW    RGET                            ; get next character
                          50    D5   0157   505          TSTL    R0                              ; is string empty?
                          16    15   0159   506          BLEQ    EXP_DONE                        ; done if true
                                015B   507 EXP_CHECK:
                          53    30   C2   015B   508          SUBL    #^A/0/, R3              ; convert to numeric
                          37    19   015E   509          BLSS    ERROR                           ; If negative, illegal character
                          09    53   D1   0160   510          CMPL    R3, #9                  ; is it a digit?
                          32    1A   0163   511          BGTRU   ERROR                           ; branch to ERROR if not
                          52    0A   C4   0165   512          MULL    #10, R2                 ; add in new digit
                          2D    1D   0168   513          BVS     ERROR                           ; overflow?
                          52    53   C0   016A   514          ADDL    R3, R2                  ; to exponent
                          28    1D   016D   515          BVS     ERROR                           ; overflow?
                          E3    11   016F   516          BRB     EXP_LOOP                        ; get more exponent digits
                                0171   517
                                0171   518 EXP_DONE:
                03 04 AE    1D    E1   0171   519          BBC     #V_NEG_DECEXP, FLAG(SP), 1$     ; check for negative
                          52    52   CE   0176   520          MNEGL   R2, R2                  ; negate DECIMAL_EXPONENT
            04 AE    10000000   8F   C8   0179   521 1$:      BISL    #M_DECEXP, FLAG(SP)     ; exponent field exists
                                0181   522
                                0181   523
```

```
                                0181    525 ;+
                                0181    526 ; Done collecting input characters for digits and/or exponent
                                0181    527 ; If FAC=0, no scaling is necessary, just store 0.0 and return.
                                0181    528 ;-
                                0181    529
                                0181    530 SCALE:
                     59   D5    0181    531         TSTL    R9                      ; Check FAC for zero.
                     1B   12    0183    532         BNEQ    INIT_BINEXP             ; Branch if not.
                                0185    533
                                0185    534 ;+
                                0185    535 ; Value is zero.
                                0185    536 ;-
                                0185    537
                                0185    538 ZERO:
                50   01   D0    0185    539         MOVL    #1, R0                  ; SS$_NORMAL
                                0188    540 ZERO_VALUE:
              51    08 AC   D0    0188    541         MOVL    value(AP), R1           ; Get address of value
              02    10 AE   91    018C    542         CMPB    DTYPE(SP), #K_DTYPE_H   ; Check length of datatype
                        02   19    0190    543         BLSS    10$
                        81   7C    0192    544         CLRQ    (R1)+
                        61   7C    0194    545 10$:    CLRQ    (R1)
                        04   0196    546         RET                             ; return with status in R0
                                0197    547
                                0197    548 ;+
                                0197    549 ; ERROR return
                                0197    550 ;-
                                0197    551
                                0197    552 ERROR:
           50   00000000'8F   D0    0197    553         MOVL    #OTS$_INPCONERR, R0     ; R0 = error return code
                        E8   11    019E    554         BRB     ZERO_VALUE              ; Set value to zero and exit
                                01A0    555
                                01A0    556 ;+
                                01A0    557 ; Set R1 to the binary exponent [exponent bias + 128 - 1].
                                01A0    558 ; 128 is number of fraction bits and 1 is
                                01A0    559 ; for the MSB fraction bit which will be hidden later.
                                01A0    560 ; BINARY_EXPONENT will be modified during normalization process.
                                01A0    561 ;-
                                01A0    562
                                01A0    563 INIT_BINEXP:
02   00   10 AE   8F    01A0    564         CASEB   DTYPE(SP), #K_DTYPE_D, #K_DTYPE_H  ; Select on datatype
                     0006'      01A5    565 1$:     .WORD   D_EXP-1$
                     000D'      01A7    566         .WORD   G_EXP-1$
                     0014'      01A9    567         .WORD   H_EXP-1$
   51   00FF 8F    3C    01AB    568 D_EXP:  MOVZWL  #^X80+^X7F>, R1         ; D-Floating
                        0C   11    01B0    569         BRB     EXP_COMMON
   51   047F 8F    3C    01B2    570 G_EXP:  MOVZWL  #<^X400+^X7F>, R1       ; G-Floating
                        05   11    01B7    571         BRB     EXP_COMMON
   51   407F 8F    3C    01B9    572 H_EXP:  MOVZWL  #<^X4000+^X7F>, R1      ; H-Floating
                        01BE    573 ;       BRB     EXP_COMMON
                                01BE    574
                                01BE    575 ;+
                                01BE    576 ; Find the true decimal exponent for the value expressed in FAC.
                                01BE    577 ; True decimal exponent = Explicit exponent - [scale factor] -
                                01BE    578 ; digits in fraction + number of overflows
                                01BE    579 ;-
                                01BE    580
                                01BE    581 EXP_COMMON:
```

```
                50   52   D0  01BE   582          MOVL    R2, R0                          ; R0 = DECIMAL_EXPONENT
                04   6C   91  01C1   583          CMPB    (AP), #<scale_factor/4>         ; is scale_factor present
                     0E   1F  01C4   584          BLSSU   20$                             ; no
          05 04 AE   06   E0  01C6   585          BBS     #V_FORCESCALE, FLAG(SP), 10$    ; force scaling
          04 04 AE   1C   E0  01CB   586          BBS     #V_DECEXP, FLAG(SP), 20$        ; ignore factor if exponent
                               01D0   587                                                  ; exists
             58   10 AC   C2  01D0   588 10$:     SUBL    scale_factor(AP), R8            ; adjust decimal exponent for
                               01D4   589                                                  ; scale factor
             58   08 AE   C2  01D4   590 20$:     SUBL    DIGITS(SP), R8                  ; adjust for digits in fraction
                               01D8   591
       0C AE   50   58   C1  01D8   592          ADDL3   R8, R0, DECEXP(SP)              ; adjust decimal exponent for overflow
                    B8   1D  01DD   593          BVS     ERROR                           ; If overflow, error
                               01DF   594
```

```
                                                                ;+
                                                                ; Normalization.  Shift the value left until bit 31 of R7 is on.
                                                                ; Adjust the binary exponent appropriately.
                                                                ;-

                09    59    D1  01DF    601          CMPL    R9, #9                  ; Are there more than 9 digits?
                      35    15  01E2    602          BLEQ    N1                      ; If not, use N1.
                12    59    D1  01E4    603          CMPL    R9, #18                 ; Are there more than 18 digits?
                      1A    15  01E7    604          BLEQ    N2                      ; If not, use N2.
                                01E9    605 ;+
                                01E9    606 ; Process all four longwords, since there are more than 18 digits.
                                01E9    607 ;-
             40 57    1F    E0  01E9    608 N4:      BBS     #31, R7, REBASE         ; Quit when R7<31> = 1.
6E     55    01    1F    EF  01ED    609          EXTZV   #31, #1, R5, TEMP(SP)   ; Save bit lost in shift.
       54    54    01    79  01F2    610          ASHQ    #1, R4, R4              ; Shift low part by one bit.
       56    56    01    79  01F6    611          ASHQ    #1, R6, R6              ; Shift high part by one bit.
56     01    00          6E  F0  01FA    612          INSV    TEMP(SP), #0, #1, R6    ; Replace bit lost in shift.
                      51    D7  01FF    613          DECL    R1                      ; Adjust exponent by one.
                      E6    11  0201    614          BRB     N4                      ; Go back and retest.
                                0203    615 ;+
                                0203    616 ; Process two low-order longwords only, since there are <= 18 digits.
                                0203    617 ;-
51     00000040    8F    C2  0203    618 N2:      SUBL    #64, R1                 ; Adjust exponent by 64.
            56    54    7D  020A    619          MOVQ    R4, R6                  ; "Shift" by 64 bits.
                  51    D7  020D    620 10$:     DECL    R1                      ; Adjust exponent by one.
       56   56    01    79  020F    621          ASHQ    #1, R6, R6              ; Shift one bit.
                  F8    18  0213    622          BGEQ    10$                     ; If R7<31> = 0, repeat.
                  54    7C  0215    623          CLRQ    R4                      ; Clear low-order 64 bits.
                  14    11  0217    624          BRB     REBASE                  ; Continue with next phase.
                            0219    625 ;+
                            0219    626 ; Process only the low-order longword, since there are <= 9 digits.
                            0219    627 ;-
51     00000060    8F    C2  0219    628 N1:      SUBL    #96, R1                 ; Adjust exponent by 96.
            57    54    D0  0220    629          MOVL    R4, R7                  ; "Shift" by 96 bits.
                  51    D7  0223    630 20$:     DECL    R1                      ; Adjust exponent.
       57   57    01    78  0225    631          ASHL    #1, R7, R7              ; Shift one bit.
                  F8    18  0229    632          BGEQ    20$                     ; If R7<31> = 0, repeat.
                  54    D4  022B    633          CLRL    R4                      ; Clear low-order longword.
                            022D    634
                            022D    635 ;+
                            022D    636 ; Rebasing.  R4-R7 now contains a binary fraction normalized with
                            022D    637 ; the radix puint to the left of bit 31 of R7.  R1 contains the
                            022D    638 ; current binary exponent and DECEXP(SP) contains the current decimal
                            022D    639 ; exponent.
                            022D    640 ;
                            022D    641 ; Therefore, the number can be represented as:
                            022D    642 ;       2**b * fraction * 10**d
                            022D    643 ; where b is the binary exponent and d is the decimal exponent.  We
                            022D    644 ; call OTS$$CVT_MUL to multiply the number by some power of 10 such
                            022D    645 ; that d goes to zero and b goes to the appropriate value.  When d is
                            022D    646 ; zero, b contains the proper binary exponent.
                            022D    647 ;-
                            022D    648
                            022D    649 REBASE:
   58    14 AE    9E  022D    650          MOVAB   BINNUM(SP), R8          ; R8 is used by subroutine as base
   28 AE    51    D0  0231    651          MOVL    R1, BINEXP(SP)          ; Store binary exponent
   14 AE    54    7D  0235    652          MOVQ    R4, BINNUM+0(SP)        ; Store fraction
```

```
                1C AE    56   7D  0239   653          MOVQ    R6, BINNUM+8(SP)
                   57    0D   D0  023D   654          MOVL    #13, R7                 ; Highest bit number possibly
                                  0240   655                                          ; on in decimal exponent.
                   52    14   D0  0240   656  10$:     MOVL    #20, R2                 ; Initially, positive offset
                50    0C AE   D0  0243   657          MOVL    DECEXP(SP), R0          ; Get decimal exponent
                         40   13  0247   658          BEQL    FLOAT                   ; If zero, we're done
                         06   14  0249   659          BGTR    20$                     ; Positive?
                   52    14   CE  024B   660          MNEGL   #20, R2                 ; No, use negative offset
                   50    50   CE  024E   661          MNEGL   R0, R0                  ; Absolute value
                   10    50   D1  0251   662  20$:     CMPL    R0, #16                 ; Within linear table range?
                         0B   15  0254   663          BLEQ    50$                     ; Yes
                03 50    57   E0  0256   664  30$:     BBS     R7, R0, 40$             ; Is the R7th bit of R0 on?
                      F9 57   F4  025A   665          SOBGEQ  R7, 30$                 ; No, try again.
                                  025D   666                                          ; This can never fall through.
             50    57    0C   C1  025D   667  40$:     ADDL3   #12, R7, R0             ; Index is 12+bit position
                                  0261   668                                          ; because table is linear
                                  0261   669                                          ; from 0-16.
                   52    50   C4  0261   670  50$:     MULL2   R0, R2                  ; Get table offset
          52   00000000'EF42   9E  0264   671          MOVAB   OTS$$A_CVT_TAB[R2], R2  ; Table entry address
                6E    57   D0  026C   672          MOVL    R7, TEMP(SP)            ; Save hi bit position
                57    28 AE   9E  026F   673          MOVAB   DECEXP+28(SP), R7       ; This is "common convert routine"
                                  0273   674                                          ; table base.  The +28 offsets
                                  0273   675                                          ; the -28 location of DEC_EXP
                                  0273   676                                          ; referenced in OTS$$CVT_MUL.
              00000000'EF   16   0273   677          JSB     OTS$$CVT_MUL            ; Do the multiplication
             57    6E    01   C3  0279   678          SUBL3   #1, TEMP(SP), R7        ; Get next bit position
                         C1   18  027D   679          BGEQ    10$                     ; Loop back if more
                                  027F   680
                                  027F   681 ;+
                                  027F   682 ; If we fall through here, then there are no more bits to reduce.
                                  027F   683 ; Test DECEXP to make sure.
                                  027F   684 ;-
                                  027F   685
          0C AE    D5   027F   686          TSTL    DECEXP(SP)                      ; Any bits still on?
             05    13   0282   687          BEQL    FLOAT                           ; No, ok
             13    19   0284   688          BLSS    UNDERFLOW                       ; Negative, underflow
           FF0E    31   0286   689          BRW     ERROR                           ; Yes, exponent too big
```

```
                     0289   691 ;+
                     0289   692 ; Create a floating number from the fraction in BINNUM and the
                     0289   693 ; binary exponent in R1.  Each datatype has a separate routine
                     0289   694 ; to do this.
                     0289   695 ;-
                     0289   696
                     0289   697
                     0289   698 FLOAT:
            28 AE    D5  0289   699          TSTL    BINEXP(SP)                      ; Underflow?
               08    19  028C   700          BLSS    UNDERFLOW                       ; Yes
02    00    10 AE    8F  028E   701          CASEB   DTYPE(SP), #K_DTYPE_D, #K_DTYPE_H
                0011' 0293   702 10$:       .WORD   FLOAT_D-10$
                0064' 0295   703          .WORD   FLOAT_G-10$
                00BD' 0297   704          .WORD   FLOAT_H-10$
                      0299   705
                      0299   706 ;+
                      0299   707 ; Value underflowed.  Check to see if it's allowed.  If so, set
                      0299   708 ; value to zero, else error.
                      0299   709 ;-
                      0299   710
                      0299   711 UNDERFLOW:
03 04 AE    02    E0  0299   712          BBS     #V_ERR_UFLO, FLAG(SP), 10$      ; Allowed?
             FEE4    31  029E   713          BRW     ZERO                            ; Yes
             FEF3    31  02A1   714 10$:     BRW     ERROR                           ; No
```

```
                          02A4   716 FLOAT_D:
      56    1C AE   7D    02A4   717          MOVQ    BINNUM+8(SP), R6        ; Restore fraction
51    28 AE   17    78    02A8   718          ASHL    #23, BINEXP(SP), R1     ; Put exponent in proper place
              45    1D    02AD   719          BVS     ERROR_D                 ; Error if overflows
           58   56    9A    02AF   720          MOVZBL  R6, R8                  ; Extract rounding bits
56    56     F8   8F    79    02B2   721          ASHQ    #-8, R6, R6             ; Shift fraction right 8 places
57  FF000000   8F    CA    02B7   722          BICL    #^XFF000000, R7         ; clear possibly shifted bits
         57      51    C0    02BE   723          ADDL    R1, R7                  ; Add in exponent
              31    1D    02C1   724          BVS     ERROR_D                 ; overflow if hidden bit bumps
                          02C3   725                                          ; exponent too far
   0B 04 AE    03    E0    02C3   726          BBS     #V_DONTROUND, FLAG(SP), 15$     ; round?
      07 58    07    E1    02C8   727          BBC     #7, R8, 15$             ; round bit is zero
              56    D6    02CC   728          INCL    R6                      ; round
           57   00    D8    02CE   729          ADWC    #0, R7
              21    1D    02D1   730          BVS     ERROR_D                 ; Error?
   04 04 AE    1B    E1    02D3   731 15$:     BBC     #V_EXT_BITS, FLAG(SP), 17$
      18 BC    58    90    02D8   732          MOVB    R8, @ext_bits(AP)
   04 04 AE    1F    E1    02DC   733 17$:     BBC     #V_NEGATIVE, FLAG(SP), 20$  ; Set sign bit
      00 57    1F    E3    02E1   734          BBCS    #31, R7, 20$            ; insert sign bit to 1
      52    08 AC    D0    02E5   735 20$:     MOVL    value(AP), R2           ; R2 = reference to result
   82    57    10    9C    02E9   736          ROTL    #16, R7, (R2)+          ; rotate and store result
   62    56    10    9C    02ED   737          ROTL    #16, R6, (R2)
            00D0    31    02F1   738          BRW     EXIT                    ; All done
                          02F4   739
                          02F4   740 ERROR_D:
            FEA0    31    02F4   741          BRW     ERROR                   ; error return
                          02F7   742
```

```
                          02F7    744 FLOAT_G:
          56    1C AE     7D    02F7    745             MOVQ    BINNUM+8(SP), R6                ; Restore fraction
    51    28 AE     14    78    02FB    746             ASHL    #20, BINEXP(SP), R1             ; Put exponent in proper place
                    4B    1D    0300    747             BVS     ERROR_G                         ; Error if overflows
 58   56    0B      00    EF    0302    748             EXTZV   #0, #11, R6, R8                 ; Extract rounding bits
      58    58      05    9C    0307    749             ROTL    #5, R8, R8                      ; Left adjust
  56    56    F5    8F    79    030B    750             ASHQ    #-11, R6, R6                    ; Shift fraction right 11 places
 57   FFE00000  8F    CA    0310    751             BICL    #^XFFE00000, R7                 ; clear possibly shifted bits
            57      51    C0    0317    752             ADDL    R1, R7                          ; Add in exponent
                    31    1D    031A    753             BVS     ERROR_G                         ; overflow if hidden bit bumps
                          031C    754                                                             ; exponent too far
       0B 04 AE     03    E0    031C    755             BBS     #V_DONTROUND, FLAG(SP), 15$     ; round?
          07 58     0F    E1    0321    756             BBC     #15, R8, 15$                    ; round bit is zero
                    56    D6    0325    757             INCL    R6                              ; round
                57  00    D8    0327    758             ADWC    #0, R7
                    C8    1D    032A    759             BVS     ERROR_D                         ; Error?
       04 04 AE     1B    E1    032C    760 15$:        BBC     #V_EXT_BITS, FLAG(SP), 17$
          18 BC     58    B0    0331    761             MOVW    R8, @ext_bits(AP)
       04 04 AE     1F    E1    0335    762 17$:        BBC     #V_NEGATIVE, FLAG(SP), 20$      ; Set sign bit
          00 57     1F    E3    033A    763             BBCS    #31, R7, 20$                    ; insert sign bit to 1
          52    08  AC    D0    033E    764 20$:        MOVL    value(AP), R2                   ; R2 = reference to result
        82    57    10    9C    0342    765             ROTL    #16, R7, (R2)+                  ; rotate and store result
        62    56    10    9C    0346    766             ROTL    #16, R6, (R2)
                0077    31    034A    767             BRW     EXIT                            ; All done
                          034D    768
                          034D    769 ERROR_G:
                FE47    31    034D    770             BRW     ERROR                           ; error return
                          0350    771
```

```
                                0350   773 FLOAT_H:
        54    14 AE    7D       0350   774          MOVQ    BINNUM+0(SP), R4        ; Restore fraction
        56    1C AE    7D       0354   775          MOVQ    BINNUM+8(SP), R6
  51    28 AE    10    78       0358   776          ASHL    #16, BINEXP(SP), R1     ; Step 1
                   69  1D       035D   777          BVS     ERROR_H                 ; Error if overflows
58      54    0F    00  EF       035F   778          EXTZV   #0, #15, R4, R8         ; Extract rounding bits
        58    58    01  9C       0364   779          ROTL    #1, R8, R8              ; Left adjust
50      56    0F    00  EF       0368   780          EXTZV   #0, #15, R6, R0         ; shift right 15 places
  54    54    F1    8F  79       036D   781          ASHQ    #-15, R4, R4
  56    56    F1    8F  79       0372   782          ASHQ    #-15, R6, R6
55      0F    11    50  F0       0377   783          INSV    R0, #17, #15, R5
 57   FFFE0000    8F  CA       037C   784          BICL    #^XFFFE0000, R7         ; clear possibly shifted bits
            57    51  C0       0383   785          ADDL    R1, R7                  ; Step 3
                  40  1D       0386   786          BVS     ERROR_H                 ; overflow if hidden bit bumps
                              0388   787                                          ; exponent too far
     11 04 AE    03  E0       0388   788          BBS     #V_DONTROUND, FLAG(SP), 15$     ; round?
        0D 58    0F  E1       038D   789          BBC     #15, R8, 15$            ; round bit is zero
                 54  D6       0391   790          INCL    R4                      ; round
                 55  00  D8   0393   791          ADWC    #0, R5
                 56  00  D8   0396   792          ADWC    #0, R6
                 57  00  D8   0399   793          ADWC    #0, R7
                  2A 1D       039C   794          BVS     ERROR_H                 ; Error?
     04 04 AE    1B  E1       039E   795 15$:     BBC     #V_EXT_BITS, FLAG(SP), 17$
        18 BC    58  B0       03A3   796          MOVW    R8, @ext_bits(AP)
     04 04 AE    1F  E1       03A7   797 17$:     BBC     #V_NEGATIVE, FLAG(SP), 20$      ; Step 4
        00 57    1F  E3       03AC   798          BBCS    #31, R7, 20$            ; insert sign bit to 1
        52    08 AC  D0       03B0   799 20$:     MOVL    value(AP), R2           ; R2 = reference to result
      82    57    10  9C       03B4   800          ROTL    #16, R7, (R2)+          ; rotate and store result
      82    56    10  9C       03B8   801          ROTL    #16, R6, (R2)+
      82    55    10  9C       03BC   802          ROTL    #16, R5, (R2)+
      62    54    10  9C       03C0   803          ROTL    #16, R4, (R2)
                              03C4   804
                              03C4   805
                              03C4   806 ;
                              03C4   807 ; Success exit
                              03C4   808 ;
                              03C4   809
                              03C4   810 EXIT:
50      01          D0        03C4   811          MOVL    #1, R0                  ; R0 = success return code
                    04        03C7   812          RET                             ; return result in @value (AP)
                              03C8   813
                              03C8   814 ERROR_H:
    FDCC          31          03C8   815          BRW     ERROR                   ; error return
                              03CB   816
```

```
                                03CB    818          .SBTTL  RGET - get next character
                                03CB    819
                                03CB    820 ;+
                                03CB    821 ; Subroutine RGET
                                03CB    822 ;       input:
                                03CB    823 ;               R0 = number of characters remaining in string
                                03CB    824 ;               R1 = address of current character
                                03CB    825 ;       output:
                                03CB    826 ;               R0 is decremented by 1.  If R0 is now non-positive,
                                03CB    827 ;               RGET returns immediately, indicating that the end
                                03CB    828 ;               of the string has been reached.
                                03CB    829 ;               If there is string remaining, R1 now points to the
                                03CB    830 ;               new current character, and R3 has that character.
                                03CB    831 ;
                                03CB    832 ;               If V_SKIPBLANKS is set in caller_flags, blanks are
                                03CB    833 ;               ignored, otherwise a blank is converted to '0'.
                                03CB    834 ;
                                03CB    835 ;               If V_SKIPTABS is set, tabs are ignored.
                                03CB    836 ;-
                                03CB    837
                                03CB    838 RGET:
                50      D7      03CB    839          DECL    R0                              ; decrement length counter
                1C      15      03CD    840          BLEQ    20$                             ; If string empty, return
                51      D6      03CF    841          INCL    R1                              ; R1 points to new character
             53 61      9A      03D1    842          MOVZBL  (R1), R3                        ; R3 gets character
 05 08 AE       04      E1      03D4    843          BBC     #V_SKIPTABS, FLAG+4(SP), 10$
                                03D9    844                                                  ; Not skipping tabs?
                                03D9    845                                                  ; FLAG is offset by 4 to allow
                                03D9    846                                                  ; for JSB to RGET.
                09      53      D1      03D9    847          CMPL    R3, #9                  ; Is it a tab?
                        ED      13      03DC    848          BEQL    RGET                    ; Yes
                20      53      D1      03DE    849 10$:     CMPL    R3, #^A/ /              ; is character a blank?
                        08      12      03E1    850          BNEQ    20$                     ; return if not
    E3 08 AE            00      E0      03E3    851          BBS     #V_SKIPBLANKS, FLAG+4(SP), RGET
                                03E8    852                                                  ; if it is a blank, and
                                03E8    853                                                  ; V_SKIPBLANKS is set, ignore
                                03E8    854                                                  ; this character.  FLAG must
                                03E8    855                                                  ; be offset by 4 to adjust
                                03E8    856                                                  ; for the JSB to RGET.
                53      30      D0      03E8    857          MOVL    #^A/0/, R3              ; set R3 to zero
                                05      03EB    858 20$:     RSB                             ; return
```

```
                        03EC   860          .SBTTL  MUL10_R9  -  multiply FAC by 10 and add digit in R3
                        03EC   861
                        03EC   862 ;+
                        03EC   863 ; Subroutine MUL10_R9
                        03EC   864 ;       input:
                        03EC   865 ;       R4-R7 - FAC
                        03EC   866 ;       R9 - count of decimal digits currently held in FAC
                        03EC   867 ;       output:
                        03EC   868 ;       R4-R7 - FAC*10 + digit in R3
                        03EC   869 ;       R9 - updated count
                        03EC   870 ;-
                        03EC   871
                        03EC   872 MUL10_R9:
          5C 59   09 F3 03EC   873          AOBLEQ  #9, R9, M1              ; If 9 or fewer digits, use M1.
             12   59 D1 03F0   874          CMPL    R9, #18                 ; If 18 or fewer digits,
                  40 15 03F3   875          BLEQ    M2                      ;  use M2.
                        03F5   876 ;+
                        03F5   877 ; Process entire octaword (four longwords), since there are > 18 digits.
                        03F5   878 ;-
                  50 DD 03F5   879 M4:      PUSHL   R0                      ; Free up a scratch register.
       50 55   01 1F EF 03F7   880          EXTZV   #31, #1, R5, R0         ; Save bit that will be lost.
          56   56 01 79 03FC   881          ASHQ    #1, R6, R6              ; Multiply high part by 2.
               56 50 C0 0400   882          ADDL    R0, R6                  ; Replace bit lost in shift.
          54   54 01 79 0403   883          ASHQ    #1, R4, R4              ; Multiply low part by 2.
       50 55   02 1E EF 0407   884          EXTZV   #30, #2, R5, R0         ; Save bits that will be lost.
          7E   56 02 79 040C   885          ASHQ    #2, R6, -(SP)           ; Multiply high part by 4.
               6E 50 C0 0410   886          ADDL    R0, (SP)                ; Replace bits lost in shift.
          7E   54 02 79 0413   887          ASHQ    #2, R4, -(SP)           ; Multiply low part by 4.
               54 8E C0 0417   888          ADDL    (SP)+, R4               ; Add 8*FAC to 2*FAC.
               55 8E D8 041A   889          ADWC    (SP)+, R5               ; ...
               56 8E D8 041D   890          ADWC    (SP)+, R6               ; ...
               57 8E D8 0420   891          ADWC    (SP)+, R7               ; ...
               54 53 C0 0423   892          ADDL    R3, R4                  ; Add digit in R3.
                  09 1E 0426   893          BCC     20$                     ; If no carry, quit now.
               55 00 D8 0428   894          ADWC    #0, R5                  ; ...
               56 00 D8 042B   895          ADWC    #0, R6                  ; ...
               57 00 D8 042E   896          ADWC    #0, R7                  ; ...
               50 8E D0 0431   897 20$:     MOVL    (SP)+, R0               ; Restore scratch register.
                  05 0434   898          RSB                             ; Return to caller.
                        0435   899 ;+
                        0435   900 ; Process two low-order longwords only, since there are <= 18 digits.
                        0435   901 ;-
       54 54   01 79 0435   902 M2:      ASHQ    #1, R4, R4              ; Multiply R4:R5 by 2.
       56 54   02 79 0439   903          ASHQ    #2, R4, R6              ; Multiply R4:R5 by 4.
               54 56 C0 043D   904          ADDL    R6, R4                  ; Add 8*FAC to 2*FAC (low).
               55 57 D8 0440   905          ADWC    R7, R5                  ; Add 8*FAC to 2*FAC (high).
               54 53 C0 0443   906          ADDL    R3, R4                  ; Add digit in R3.
               55 00 D8 0446   907          ADWC    #0, R5                  ; ...
                  56 7C 0449   908          CLRQ    R6                      ; Restore R6:R7.
                  05 044B   909          RSB                             ; Return to caller.
                        044C   910 ;+
                        044C   911 ; Process low-order longword only, since there are 9 or fewer digits.
                        044C   912 ;-
          54 6444   DE 044C   913 M1:      MOVAL   (R4)[R4], R4            ; Multiply R4 by 5.
          54 6344   3E 0450   914          MOVAW   (R3)[R4], R4            ; Multiply R4 by 2 and add R3.
                02  12 0454   915          BNEQ    10$                     ; If nonzero, quit now.
                59  D4 0456   916          CLRL    R9                      ; Reset digit count, since digit
```

```
                0458   917
                0458   918 10$:    RSB                             ;  was not significant.
                0459   919                                         ; Return to caller.
                0459   920          .END
```

```
BINEXP          =  00000028
BINNUM          =  00000014
CALLER_FLAGS    =  00000014
CHECK_DIGIT        000000A7  R      01
COMMON             0000002B  R      01
CRY             =  00000040
DECEXP          =  0000000C
DECIMAL_POINT      0000011E  R      01
DIGITS          =  00000008
DIGITS_IN_FRACT =  0000000C
DIGIT_LOOP         0000009D  R      01
DTYPE           =  00000010
D_EXP              000001AB  R      01
ERROR              00000197  R      01
ERROR_D            000002F4  R      01
ERROR_G            0000034D  R      01
ERROR_H            000003CB  R      01
EXIT               000003C4  R      01
EXPON              00000129  R      01
EXP_CHECK          0000015B  R      01
EXP_COMMON         000001BE  R      01
EXP_DONE           00000171  R      01
EXP_LOOP           00000154  R      01
EXP_MINUS          00000116  R      01
EXP_NEG            0000014C  R      01
EXP_PLUS           0000010E  R      01
EXT_BITS        =  0000001B
FLAG            =  00000004
FLOAT              00000289  R      01
FLOAT_D            000002A4  R      01
FLOAT_G            000002F7  R      01
FLOAT_H            00000350  R      01
FOR$CNV_IN_DEFG    0000001E  RG     01
FRAME           =  00000050
G_EXP              000001B2  R      01
H_EXP              000001B9  R      01
INIT_BINEXP        000001A0  R      01
IN_STR          =  00000004
K_DTYPE_D       =  00000000
K_DTYPE_G       =  00000001
K_DTYPE_H       =  00000002
L_2P31_DIV_10   =  0CCCCCCC
M1                 0000044C  R      01
M2                 00000435  R      01
M4                 000003F5  R      01
MUL10_R9           000003EC  R      01
M_DECEXP        =  10000000
M_DEC_POINT     =  40000000
M_DONTROUND     =  00000008
M_EXT_BITS      =  08000000
M_NEG_DECEXP    =  20000000
N1                 00000219  R      01
N2                 00000203  R      01
N4                 000001E9  R      01
NOT_DIGIT          000000C9  R      01
NO_OF_FLAGS     =  00000007
OTS$$A_CVT_TAB     ********    X    00
OTS$$CVT_MUL       ********    X    00
OTS$CVT_T_D        0000001E  RG     01
OTS$CVT_T_G        0000000F  RG     01
OTS$CVT_T_H        00000000  RG     01
OTS$_INPCONERR     ********    X    00
REBASE             0000022D  R      01
REGMASK         =  000003FC
RGET               000003CB  R      01
SCALE              00000181  R      01
SCALE_FACTOR    =  00000010
TEMP            =  00000000
UNDERFLOW          00000299  R      01
VALUE           =  00000008
V_DECEXP        =  0000001C
V_DEC_POINT     =  0000001E
V_DONTROUND     =  00000003
V_ERR_UFLO      =  00000002
V_EXP_LETTER    =  00000005
V_EXT_BITS      =  0000001B
V_FORCESCALE    =  00000006
V_NEGATIVE      =  0000001F
V_NEG_DECEXP    =  0000001D
V_ONLY_E        =  00000001
V_SKIPBLANKS    =  00000000
V_SKIPTABS      =  00000004
ZERO               00000185  R      01
ZERO_VALUE         00000188  R      01
```

Psect synopsis

PSECT name	Allocation	PSECT No.	Attributes
. ABS .	00000000 (0.)	00 (0.)	NOPIC USR CON ABS LCL NOSHR NOEXE NORD NOWRT NOVEC BYTE
_OTS$CODE	00000459 (1113.)	01 (1.)	PIC USR CON REL LCL SHR EXE RD NOWRT NOVEC LONG

Performance indicators

Phase	Page faults	CPU Time	Elapsed Time
Initialization	32	00:00:00.05	00:00:02.00
Command processing	123	00:00:00.31	00:00:03.29
Pass 1	102	00:00:01.46	00:00:05.28
Symbol table sort	0	00:00:00.05	00:00:00.05
Pass 2	163	00:00:01.07	00:00:05.44
Symbol table output	8	00:00:00.07	00:00:01.42
Psect synopsis output	2	00:00:00.01	00:00:00.01
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	432	00:00:03.02	00:00:17.49

The working set limit was 1200 pages.
16746 bytes (33 pages) of virtual memory were used to buffer the intermediate code.
There were 10 pages of symbol table space allocated to hold 87 non-local and 37 local symbols.
920 source lines were read in Pass 1, producing 19 object records in Pass 2.
0 pages of virtual memory were used to define 0 macros.

Macro library statistics

Macro library name	Macros defined
_255DUA28:[SYSLIB]STARLET.MLB;2	0

0 GETS were required to define 0 macros.

There were no errors, warnings or information messages.

MACRO/ENABLE=SUPPRESSION/DISABLE=(GLOBAL,TRACEBACK)/LIS=LIS$:OTSCVTTR/OBJ=OBJ$:OTSCVTTR MSRC$:OTSCVTTR/UPDATE=(ENH$:OTSCVTTR)

OTSCVTPD
LIS

OTSCVTTF
LIS

OTSCVTRFP
LIS

OTSCVTRT
LIS

OTSCVTTOL
LIS

OTSCVTPG
LIS

OTSCVTTR
LIS

OTSCVTRHP
LIS

OTSLINKAG
LIS

OTSCVTRDP
LIS

OTSCVTTIL
LIS

OTSCVTTLL
LIS

OTSCVTPF
LIS

OTSCVTRGP
LIS

OTSMOVE
LIS

OTSMSG
LIS

OTSCVTPH
LIS

OTSLUN
LIS